Flexible use of IP Cores on Dynamically Reconfigurable Systems

Miguel L. Silva
DEEC, Faculdade Engenharia
Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 PORTO, Portugal
mlms@fe.up.pt

João Canas Ferreira
INESC Porto,
Faculdade Engenharia
Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 PORTO, Portugal
jcf@fe.up.pt

José Silva Matos
INESC Porto,
Faculdade Engenharia
Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 PORTO, Portugal
jsm@fe.up.pt

***Abstract*—The advantages of dynamic reconfiguration can only be exploited if devices, tools and design flows are available to support the partial reconfiguration of FPGA-based systems. For a number of applications, enabling the swap of cores at run-time, under software control, is an essential feature that allows tailoring the system response to the needs of different methods, standards and power/performance requirements. The paper proposes a method to support the exchange of intellectual property (IP) cores during system operation. The approach is based on the definition of a base system, with reserved or dynamic areas, where different cores may be plugged in, providing time-sharing of the system resources. It is shown how bitstream-level IP cores can be used in a design flow that allows different cores to be used in one or more host areas, with minimal intervention from the designer. A demonstration system along with example applications are presented to illustrate the approach.**

I. Introduction

Designers of complex reconfigurable systems have often relied on pre-designed intellectual property (IP) cores to attain good performance within the bounds of acceptable development times. Both programmable core generators from FPGA providers and individual IP cores from third parties are available. Typically, hard IP cores are delivered as netlists of low-level FPGA building blocks, together with VHDL or Verilog wrappers and simulation models, and are imported into the regular design flow supported by standard development tools. The use of IP cores from multiple sources in dynamically reconfigurable systems [1] is impractical today, due to the lack of support for swapping cores with different physical dimensions and terminal positions during system operation (at run-time). Such a capability would allow the running system to adapt precisely to the applications needs at any given time. For instance, a system might support different encryption methods or multiple communication standards without the need to have hardware dedicated to each supported alternative simultaneously on chip. The paper tackles the problem of providing support for IP core swapping. It considers the situation where a region of the reconfigurable circuit is reserved as a "dynamic" area, with a fixed interface to the rest of the system. IP cores can be loaded to this area on the fly under software control

First author funded by FCT scholarship SFRH/BD/17029/2004.

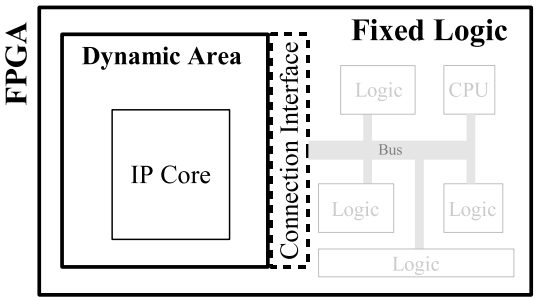

Fig. 1. Generic system organisation.

(see figure 1). The software may be running on an embedded processor or on an external CPU, and the reconfigurable device must support partial reconfiguration. The tools described in this paper target the Virtex-II Pro family from Xilinx, but similar approaches are applicable to other device families.

The use of IP cores in dynamically reconfigurable designs requires solving the problem of adapting the IP core to the dynamic area. Requiring all exchangeable IP cores to have exactly the same interface (i.e., the same types of terminals at the same physical positions) would severely limit the user's choice. Instead, we choose to adjust the IP core to the dynamic area, by creating the required interconnections to the dynamic area interface as needed. In this way, IP cores with different interfaces at the physical level can use the same dynamic area. The IP core is still required to fit in the dynamic area, which should be sized for the largest core to be used.

The method described in this paper is built around a tool that takes partial bitstreams of IP cores (plus some additional information) and generates a new partial bitstream that can be used to configure a predefined dynamic area. The new bitstream includes any connections necessary to adapt the core to the dynamic area interface. This means that cores with different physical layouts can be "inserted" in the same dynamic area. The use of reserved areas (or slots) for hosting different circuits at run-time is standard: the main issues are discussed in [2], together with an extended example. The exploitation of partial dynamic reconfiguration in the context

of car electronics (but with wider applicability) is described at length in [3].

The concept of bitstream-based IP cores has been treated previously in [4] where IP cores are merged with the base design and are required to have matching terminals at specific positions. The whole design is then loaded to the FPGA. Run-time reconfiguration is not considered. In [5] [6] [7] the use of IP Cores on a dynamically reconfigurable system is tackled: each work presents a solution for a communication interface between the reconfigurable IP Cores and the rest of the system. They have in common the restriction of requiring matching terminals at specific positions. In our approach the designer creates a base system with one or more dynamic areas for dynamic reconfiguration using standard design tools. The interface between dynamic areas and the base system included in this design is fixed. Our bitstream manipulation tool is then used to create the bitstreams used for partial reconfiguration of the dynamic areas.

This approach has the important advantage of allowing the base design to be used with cores from different providers, and even with cores that become available after the base design is finished. Furthermore, providing design information in bitstream format increases the level of protection for the intellectual property involved. In addition, the process allows easy integration of watermarking techniques for intellectual property protection [8].

A relevant aspect of this approach is its capability to allow connecting together two or more IP cores in the same dynamic area. This is useful for systems that comprise several stages that can be used in different combinations. Instead of having as many different cores as possible combinations of the different stages, it is more flexible to provide several compatible cores and let the designer generate the required combinations as necessary for a given project. Again, it is not necessary for the cores to have compatible layouts, since the interconnections can be generated as required. Our tool is also capable of handling this task.

The rest of the paper is organised as follows: Section II elaborates on the use of IP cores for dynamic reconfiguration, the associated opportunities and requirements, and the features of the proposed approach. Section III discusses how IP cores may be used from a designer's perspective. Design flow issues area addressed, as well as run-time management requirements. Section IV provides mores details on the automatic tool that processes the bitstreams of IP cores and adapts them to the dynamic area. Restrictions on the architecture of the core imposed by the bitstream manipulation process are also addressed there. Section V presents the implementation of a simple base system on a Virtex-II Pro FPGA and examples of IP-core-based dynamic reconfiguration applications running on the system. Section VI presents some concluding remarks and closes the paper.

II. Bitstream-level IP cores

The present paper is concerned with the use of run-time-exchangeable IP cores for dynamically reconfigurable systems. It is assumed that the system's logic remains unchanged during execution, except for an area of the FPGA, which is reserved for components that are loaded at run-time (see figure 1). The loading and unloading of components occurs under control of a program, possibly running on an embedded processor. The FPGA is assumed to support partial reconfiguration. This setup is easily implemented with platform FPGAs like the Virtex-II Pro from Xilinx (or other current or future FPGA families with similar characteristics). Alternatives with more than one dynamic area, specially designed configuration controllers or, even, reconfiguration of whole chips may be envisioned, but do not appreciably change the issues addressed here. The discussion is framed in terms of a Virtex-II-Pro-based system, because that is what was used in our proof-of-concept implementation.

Under the arrangement described, the interface between dynamic area and base system must remain fixed. It may be connected to the processor bus or it may be part of a dedicated communication channel.

In the simplest use scenario, different IP cores are loaded (non-simultaneously) onto the dynamic area, effectively time-sharing its logical resources. The IP cores must be compatible, in the sense that they must have similar inputs and outputs that functionally match the interface of the dynamic area. For instance, if the dynamic area interface implements a simple connection to a 32-bit bus, the IP core may have up to 32 inputs and 32 outputs, and must operate within the timing constraints of the bus. Which IP core is in use at a given time may be determined by the application according to the current inputs; alternatively, the cores may be exchanged according to some predefined schedule.

Dynamic reconfiguration may be used to exchange functionally similar cores in many situations. Application areas where support for multiple implementations is interesting include encryption and digital message signing, data compression, signal filtering and video acquisition. Alternatively, it may be advantageous to have different version of the same basic function with different implementation tradeoffs (e.g., for power management purposes).

In any case, requiring that the IP core's terminals match the physical terminals of the dynamic area interface imposes a severe restriction on the core, limiting the opportunities for re-utilisation. The approach proposed here to alleviate such a stringent requirement is to let the designer adapt the terminals of the IP core to the dynamic area interface. This can be done by creating connections between both sets of terminals. The automatic tool developed by the authors can take a specification of a dynamic area (its dimensions and the position of the terminals) together with the partial bitstream defining an IP core (plus information on the terminals) and produce a new partial bitstream that combines the IP core and the new interconnections (see figure 2). In the process, the core's bitstream is relocated to the position of the dynamic area. The new bitstream also ensures that the whole dynamic area is correctly configured (i.e., any previous contents are effectively erased).

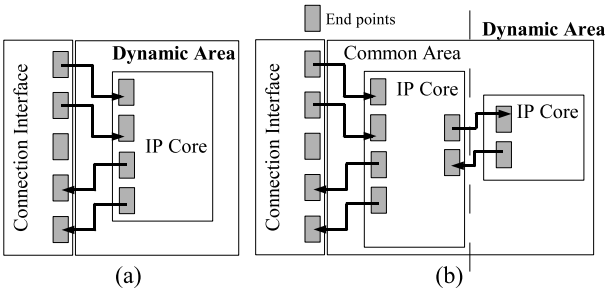

Fig. 2. Examples of using different cores in the same area. Case b) illustrates a setup where one core may be common to several different configurations.

A second, more complicated scenario considers combining two or more cores in the same dynamic area. In order to illustrate the usefulness of the concept, consider the problem of providing hardware support for encryption. NIST standards define 3 different block cipher algorithms (AES, Triple DES and Skipjack) which can be used in 7 modes of operation. Instead of providing 21 different cores implementing all the possible combinations, it is possible to provide the 10 building blocks and let the designer generate the required combinations for a specific application.

In this case, again, it is too restrictive to demand that all the cores have matching terminal positions. The solution is to expand the "adaptation" required in the first scenario to full-blown "linking" of different cores (see figure 2b)). The tool described in this work is capable of carrying out the placement and routing necessary to achieve this goal. In this scenario, the IP provider may deliver a complete line of logically compatible cores, which can be combined as needed by the system designer.

The greatest limitation of the approach just described is that it does not contemplate the inclusion of additional "glue logic" that may be needed for functionally adapting cores that were not designed with the intent of being combined. A possible solution is to allow the designer to create his/her own small core implementing the needed glue logic functionality, and then to combine it with the other IP cores. Since the procedures for designing IP cores use the same tools as the standard design process (see Section IV), it is easy to define a streamlined design process for creating small components with simple timing constraints.

III. System design with bitstream-level IP cores

This section describes the process of creating a base design for a system that takes advantage of dynamically reconfigurable bitstream-level IP cores. The base design will typically consist of two sections: the static section, which will remain unchanged throughout the application's lifetime, and the dynamic section that is time-shared between IP cores at run-time. The dynamic section consists of one or more reserved areas (dynamic areas) that must take into account the characteristics of the different IP cores that will be used: besides the size of the areas, the relative positions of the basic FPGA resources (configurable logic blocks, RAM blocks, etc.) have to be considered. The reserved areas must be completely empty: no logic resources or interconnections in the area can be used in the base design.

In addition, the designer has to account for the architectural limitations of the specific platform in use. As an example, devices of the Virtex-II Pro family (like other Xilinx devices) are reconfigured by frames. A frame is a set of configuration bits that control a column of configurable resources. Each such column covers the entire height of the device. Therefore, the reconfigurable area would ideally have a rectangular shape, extending over the full height of the device. However, in some practical situations this can be difficult to achieve. In effect, even allowing for rectangular shapes that only occupy part of the device's height may not be enough, and it may be necessary to use several piecewise rectangular areas.

There are two main reasons for this situation. First, the internal architecture of the FPGA is not completely homogeneous, with some areas having specific modules. For example, Virtex-II Pro devices may have several PowerPC cores that "interfere" with the area available for configuration.

A second problem is presented by design constraints, which may include constraints on the position of the primary input/output connections (used to connected to external components like DRAM and communication ports), or internal restrictions deriving from the use of specific modules like the ICAP (Internal Configuration Access Port) or JTAG interface. The demonstration system described in section V illustrates the use of piecewise rectangular areas.

Whereas the preceding discussion illustrates device-specific limitations, it is always necessary to pay close attention to the interface between the fixed region and the dynamic area. For Xilinx devices, the most common approach is to use a hard macro, a block that has been previously routed and placed [9]. The use of such a "bus macro" avoids the need for routing and allows a clear separation between interconnection and logic at design time. The original implementation of bus macros is described in [9]. In that approach, the communication between IP cores and the base system is done through unidirectional long lines connected to the output of tri-state buffers. An alternative approach is to use LUTs for the implementation of the fixed-point connections [10] (see figure 3). Bus macros can be thought of as fixed connections that are split: half of the connection belongs to the fixed system and the other half must be included in the core. At run-time, the core is placed on the device in such a way that its part of the interface abuts the corresponding part of the core's bus macro. In this approach, IP cores must have the same connection interface as the system design and the re-usability of the IP cores is reduced.

In order to have a more flexible solution, the proposed approach is centered around a bitstream manipulation tool that is used to place a core inside a reserved area and to create the necessary interconnections. This is done using the bitstream of the base system, the partial bitstream of the core and a specification of the connection endpoints (in the dynamic area interface and in the IP core) to create a partial bitstream adapted to the specific dynamic area (see Section IV).

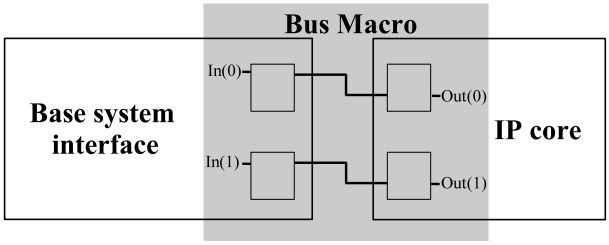

Fig. 3. A bus macro: half of the bus macro belongs to the IP core and half belongs to the base system interface (or to another core).

The connection endpoints are defined as either inputs or outputs, and the number of endpoints of each type defined in the base design has to be compatible with the IP core interface. The interfaces do not have to be the same: some endpoints on either interface may be left unconnected (or, in the case of input terminals, connected to constant logic values) as long as this is functionally acceptable.

For Xilinx devices the base system design can be done with the tools provided by the vendor. These support a design flow called "Modular Design" [11], whose main objective is to reduce the overall development time by constructing a complete design from independently developed modules. The same design flow can be used to create designs with reserved areas for the IP cores. The main steps of the design flow are the following:

- Specification of the base design, using any supported method (i.e. Verilog, VHDL , EDK etc.)
- Definition of the connection interface, including the definition of input and output terminals (position and number).
- Dimensioning of the reserved area by specification of an appropriate set of constraints to avoid placement and routing in the reserved areas.
- Generation of the bitstream for the base design (after mapping, placement and routing).

A demonstration base system is described in section V.

IV. Bitstream assembly

Our bitstream-processing tool does the integration of the design with the IP Cores. It was originally developed to automatically build full or partial bitstreams from the bitstreams of individual components and is targeted at Virtex-II-Pro-based reconfigurable systems [12]. The program is written in Java and uses the JBits library [13] for low-level bitstream processing tasks. It can produce partial bitstream for dynamic reconfiguration, as well as full bitstreams combining the base system and IP cores. The last option allows the construction of complete, static designs from a base system and several IP cores for situations where dynamic reconfiguration is not necessary. The tool takes as input the bitstream of the base system and the partial bitstreams of the IP cores. Additional information about the connection points must also be provided by the user. The following is a summary of the functions that can be performed:

- The IP cores can be relocated from their initial position to compatible areas.
- Multiple IP cores can be used in same design and the same core can be used multiple times.
- Area compatibility tests between the IP cores and the desired locations are carried out.
- The tool creates the connections between an IP core and the base design, and between several IP cores. This involves placing the IP cores in the dynamic and automatically routing the interconnections between them.

The user starts by specifying the bitstreams for the base system and for all IP cores. For each core, the user may define a location or may allow the program to perform the placement. The program checks if the core's destination area is suitable (it must have the same logic resources as the original area). After the placement steps, the interconnections are routed, if necessary (some cores may just connect by abutment). The routing algorithm performs a breadth-first, exhaustive search to establish the route for each successive connection. This ensures that the shortest path for each connection is found, but it does not ensure that a global optimum for all connections is obtained. Placement or routing may fail if not enough resources are available in the dynamic area. The user can make manual adjustments to the placement to facilitate another possible route.

For the work described here, the tool was extended to handle piecewise rectangular regions, possibly connected by "bridges". The previous version could only handle single rectangular dynamic areas, whereas the newest version supports multiple areas of different piecewise rectangular shapes.

The final stage of the process builds the output bitstream. This may be either a full bitstream for the whole assembly or a partial bitstream suitable for reconfiguring the dynamic area (without affecting any other circuitry above or below the target area).

In general, several different partial bitstreams must be produced, each corresponding to a possible combination of IP cores. For each one, the above process must be repeated.

A slight optimisation is possible if the partial configurations have common sections (e.g., a multi-core setup where one core appears in all configurations, see figure 2b). In this case, it is possible to first extend the base system by merging in a given core's partial bitstream. The remaining cores are then placed and routed with respect to that extended base system.

The IP cores can be created with the standard design tools provided by the vendor; no further tools are required. However, some issues must be taken in consideration in the process. The most important one is that inputs and outputs of the core must belong to the configurable logic blocks on the periphery of the core, since our tool does not inspect the interior of the cores. The core designer should also note that using dedicated resources like block RAMs and multiplier blocks imposes more severe restrictions on the placement of the core. Core designs must be restricted to a specific area of the device by specifying the appropriate constraints to the vendor tools. The exact position of the area is not relevant, as the core

will be relocated as needed. After the final bitstream has been generated, the partial bitstream corresponding to the specific core must be extracted. In our case, a special version of the tool is used to extract the partial bitstream together with some additional information that is required for later use of the core. The information about a core is stored in a file that contains, in addition to the bitstream data, the following items of information:

- size of the core (width and height);
- relative position of input and outputs;
- clock lines and clock frequency.

This file may be encrypted in order to provide extra security when deploying the IP core.

V. A Basic Demonstration System

This section presents a simple base system and demonstrates a possible use of dynamically reconfigurable IP cores. The demonstration system is built on a development board (XUP Virtex-II Pro Development System) equipped with a Xilinx XC2VP30-7 FPGA and 512 MB of external memory (DRAM). The FPGA contains two embedded PowerPC (PPC) 405 processor cores, but only one is used.

The static area of the system contains the following modules:

- A memory interface unit: the external memory is used to store both the reconfiguration data for dynamic modules and application-specific data.
- A reconfiguration control unit that performs reconfigurations using the Virtex-II Pro Internal Configuration Access Port (ICAP).
- A serial communications unit responsible for transferring data to/from external devices (for instance, a controlling computer).
- A dynamic area control unit that supports the communication with the modules in the dynamic area. It includes a DMA controller, two input FIFOs and one output FIFO.
- A sound controller that enables communication to the on-board AC'97 audio codec.

The Xilinx Embedded Development Kit (EDK) was used for the first stage of system design. The implementation of the design was made using Xilinx Integrated Software Environment(ISE) and Xilinx PlanAhead.

As shown on Fig. 4, the choice was made to concentrate the majority of the static design on the left site of the FPGA, next to the external connection to the DRAM. Two channels were created, in order to connect these modules to the ICAP and JTAG port. The bottom channel is used for the connection to ICAP and the channel above the CPU occupied by the connections to the JTAGPPC module (and the JTAG port). The JTAG connection to the unused CPU is necessary for the overall JTAG-based debug infrastructure to work properly.

The audio control unit is also placed on the right hand side of the FPGA, near to the pins that are connected to the external AC'97 codec. This unit is not directly connected to rest of the static design, but will rather work properly only when the

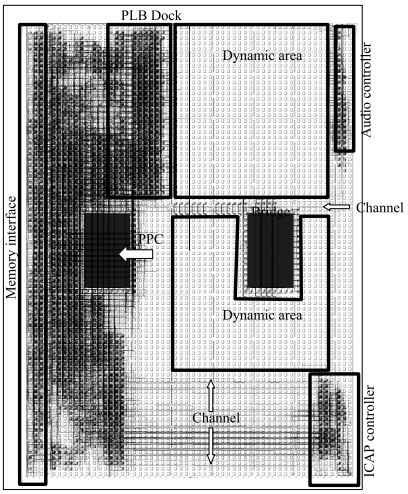

Fig. 4. Floorplan of base system with piecewise rectangular dynamic area.

required logic is present in the dynamic area. The final system design has a dynamic area divided in two parts, one with a rectangular shape and the other with a U-like shape.

A "bridge" was included in the base system in order to connect the two regions of the dynamic area. The connection interface used to build this bridge is based on the same principles used to define the connection interface between the static an dynamic area.

The general characteristics of the system are as follows. The CPU operates at 300 MHz, and the processor local bus (PLB) and on-chip peripheral bus (OPB) use a 100 MHz clock. All the measured timings reported in this paper were obtained with this setup. The audio controller configured in the static area is connected to an external chip (National Instruments LM4550) present on the development board. The LM4550 implements revision 2.1 of the AC'97 codec specification. The controller accepts 32-bit-wide input data, corresponding to left and right 16-bit PCM audio channels, and produces audio data in the same format when recording external audio signals.

For the current example, the following individual IP cores were developed for use in the dynamic area:

- A tone generator that can be configured to different frequencies. This component has two stages: one contains the signal sample and the other one sets the output frequency by controlling the periodic sweep of the samples.
- Three different 16-bit filters: band-pass, low-pass, and high-pass filters.
- An audio input controller with a FIFO for the storage of input sound samples.

The IP cores may be combined in several ways (see Fig. 5). The following three simple example applications were implemented and tested:

- A tone generator is loaded to the dynamic area and then reconfigured to generate tones at a different frequency, by changing the second stage.
- An audio sample generated by the CPU (from data kept in the external DRAM) is sent through two different filters

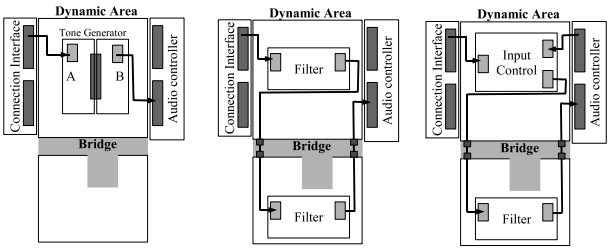

Fig. 5. Placement of components for RTR assemblies.

TABLE I
ASSEMBLY TIMES AND NUMBER OF PROCESSED INTERCONNECTIONS FOR THE THREE EXAMPLES OF FIG. 5

	Number of connections	Assembly time routing (s)
Example 1	48	3,5
Example 2	104	7,3
Example 3	72	5,3

connected in series.

- The audio input controller is used to record an external source; the resulting stream is passed through a filter and sent to the output.

Figure 5 shows the placement and the main connections flow for the three examples. The filter blocks may be any of the three available ones.

For these examples the tool placed the components as expected (matching the positions in Fig. 5) and generated the partial bitstreams for the corresponding RTR assemblies. A simple driver application, written in C, allows the user to switch between the available RTR assemblies.

The assembly time taken for each example is shown in Tab. I, together with the number of interconnections processed. The times were measured on a PC with a 1.6 GHz AMD Athlon and 512 MB of main memory using the J2SE 1.4 Java distribution. The number of frames is the same for all generated assemblies (528 frames) and, hence, the reconfiguration time is also the same (7.5 ms).

In the first example, partial reconfiguration is also used to change one of the stages of the tone generator, thereby changing the frequency of the generated tone. Three different implementations of that stage B are available. Therefore, our tool can be used to create three additional bitstreams, that can be used to reconfigure only the corresponding part of the dynamic area. These bitstreams are smaller (132 frames) and the associated reconfiguration takes only 1.9 ms.

The assembly times are very small compared with the time that would be necessary to produce the partial bitstream using the traditional design flow. The generated assemblies functioned correctly when loaded to the development board.

VI. CONCLUSION

This paper addressed the problems of swapping IP cores in dynamically reconfigurable systems. In contrast to the traditional approach that requires the IP cores to match exactly the terminal positions of the base system interface (thus strongly restricting the reuse of IP cores from different sources), we propose a solution centred around a tool that places an IP core in a reserved area of the dynamic system and routes all the required interconnections. This tool is also able to connect together multiple cores in the same area. This capability can be used to put together systems from two or more cores and to include any "glue logic" required. The tool only requires the bitstreams of the base system and the partial bitstreams of the IP cores, plus information on the relative positions of the terminals to be connected. All other tasks required to put together a system or to create IP cores can be performed using standard vendor tools. The fact that only raw bitstreams are required for the assembly process may be used to increase the protection against IP loss.

The paper closes includes a small demonstration of the proposed approach, illustrating its flexibility and the results that can be obtained by its application.

REFERENCES

[1] Katherine Compton and Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.

[2] Pete Sedcole, Brandon Blodget, Tobias Becker, James Anderson, and Patrick Lysaght. Modular dynamic reconfiguration in virtex fpgas. *IEE Proceedings Computers & Digital Techniques*, 153(3):157–164, May 2006.

[3] J. Becker, M. Hübner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka. Dynamic and partial FPGA exploitation. *Proceedings of the IEEE*, 95(2):438–452, 2007.

[4] Edson L. Horta and John W. Lockwood. Automated method to generate bitstream intellectual property cores for Virtex FPGAs. In *Field Programmable Logic and Applications (Proc. FPL'04)*, volume 3203 of *Lecture Notes in Computer Science*, pages 975–979. Springer, 2004.

[5] J.C. Palma, J.C. Palma, A.V. de Mello, L. Moller, F. Moraes, and N. Calazans. Core communication interface for fpgas. In A.V. de Mello, editor, *Proc. 15th Symposium on Integrated Circuits and Systems Design*, pages 183–188, 2002.

[6] H. Kalte, H. Kalte, D. Langen, E. Vonnahme, A. Brinkmann, and U. Ruckert. Dynamically reconfigurable system-on-programmable-chip. In D. Langen, editor, *Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 235–242, 2002.

[7] Yana E. Krasteva, Ana B. Jimeno, Eduardo de la Torre, and Teresa Riesgo. Straight method for reallocation of complex cores by dynamic reconfiguration in Virtex II FPGAs. In *Proc. 16th IEEE Int. Workshop Rapid Syst. Prototyp. (RSP'05)*, pages 77–83, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[8] John Lach, William H. Mangione-Smith, and Miodrag Potkonjak. Robust fpga intellectual property protection through multiple small watermarks. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 831–836, New York, NY, USA, 1999. ACM.

[9] Xilinx. Two flows for partial reconfiguration: Module base or small bit manipulations. Application note 290, September 2004.

[10] Michael Hübner, Tobias Becker, and Jürgen Becker. Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration. In *Proc. SBCCI '04*, pages 28–32, New York, NY, USA, 2004. ACM Press.

[11] Xilinx. *Development System Reference Guide*, 2005.

[12] Miguel L. Silva and João Canas Ferreira. Generation of hardware modules for run-time reconfigurable hybrid CPU/FPGA systems. *IET Comput. Digital Tech.*, 1(5):461–471, 2007.

[13] Steven A. Guccione and Delon Levi. XBI: A Java-based interface to FPGA hardware. In John Schewel, editor, *Configurable Computing: Technology and Applications*, volume Proc. SPIE 3526, pages 97–102, Bellingham, WA,USA, November 1998. SPIE.